Low Overhead Real-Time Computing with General Purpose Operating Systems

Mr. Michael A. Raymond
Silicon Graphics Inc.
Phone: (651) 683-3434
Fax: (651) 683-5098
Email: mraymond@sgi.com

This topic addresses the Software Architecture and Case Study of HPEC categories.

Abstract: Much embedded real-time computing is done with operating systems crafted from the ground up specifically for the task. In larger systems and more recently, general-purpose operating systems such as SGI IRIX and Linux are used for new projects because they already have multiprocessor and device driver support as well a large user base. Work can and has been done to improve their real-time capabilities, for example IRIX has been improved over the years to offer a number of hard real-time response time guarantees ranging from 1ms to 50us. Since general-purpose operating systems must meet the needs of a wide variety of users they are very complex which can interfere with and cause variability in the most demanding of real-time applications. My presentation deals with one method of improving the performance of these operating systems, which is by adding special low overhead interfaces to common facilities, which may be accessed entirely in user space. I will describe a number of these features including low overhead priority inversion avoiding locking methods and device I/O as well as provide performance comparisons.

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
20 AUG 2004	**N/A**	**-**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Low Overhead Real-Time Computing with General Purpose Operating Systems	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **Silicon Graphics Inc.**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release, distribution unlimited

13. SUPPLEMENTARY NOTES
See also ADM001694, HPEC-6-Vol 1 ESC-TR-2003-081; High Performance Embedded Computing (HPEC) Workshop (7th)., The original document contains color images.

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**UU**	**15**	

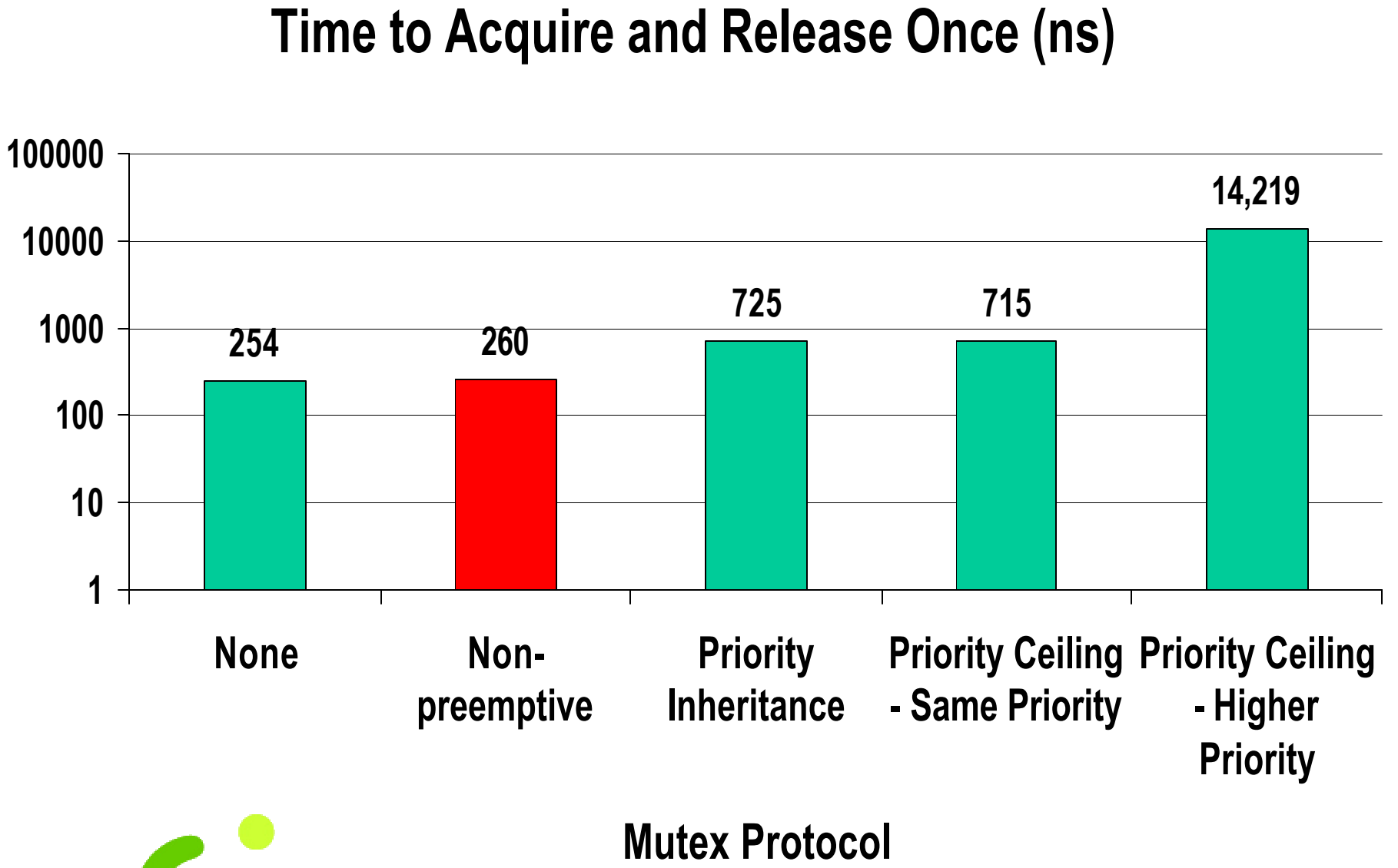
Low Overhead Real-Time Computing

- *General Purpose OS's can be highly unpredictable*
  - Linux response times seen in the 100's of milliseconds
- *Work around this by isolating from interrupts and other processes and kernel threads*
- *Put as much functionality into user space as possible*

User Space Functionality

- *Memory map hardware and clocks, and handle interrupts in user space*
- *Use shared memory for IPC*
- *Memory map special kernel pages or use virtual system calls*
- *Do uncontended locking with IRIX usyncs or Linux futuxes*

Mutex Protocol Performance



Overview

- *General Purpose OS's are complicated and unpredictable*

- *Good RT performance may be had by protecting cpu's from outside interference*

- *User level interfaces to common features further reduce latency and unpredictability*

- *Summary: Lock out and avoid as much of the kernel as possible*

Methods of Blocking Interruptions

- *Restrict cpu's to specific threads*
- *Isolate cpu's from TLB and cache invalidations*
- *Direct hardware interrupts elsewhere*
- *Turn off the timeshare scheduling interrupt*
- *Ward off any remaining kernel background tasks*

Lock in User Space When Possible

- *IRIX? usyncs and Linux? futexes attempt to handle most locking actions with atomic primitives outside the kernel*

- *Contended locks force accessors into the kernel to use wait queues*

- *Usually just costs a few instructions vs entering and leaving the kernel for a free lock*

SGI? IRIX? Non-preemptive Mutexes

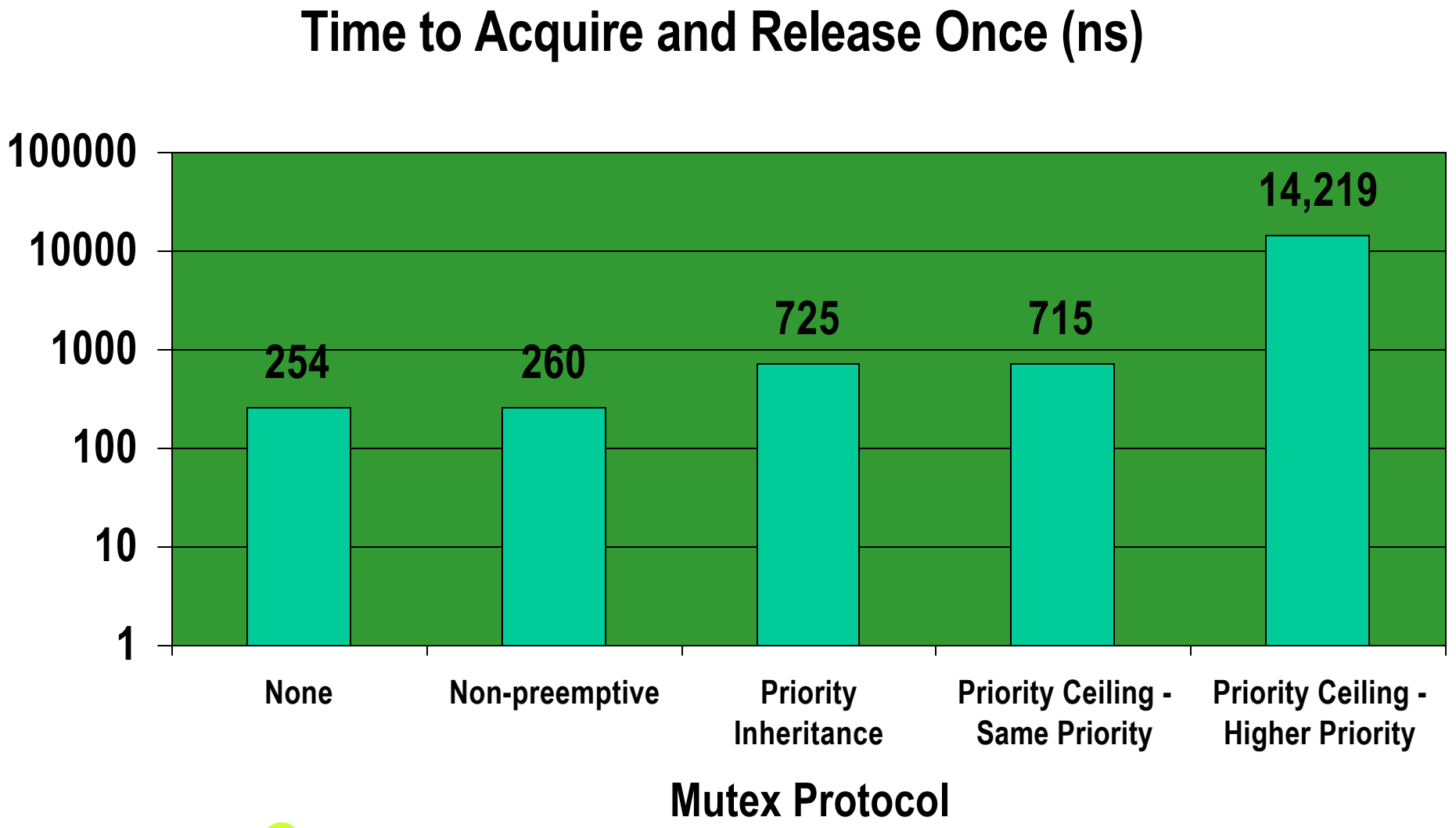
- *Priority Inheritance and Priority Ceiling both have overhead even with uncontended locks*

- *IRIX non-preemptive mutexes provide pseudo - priority ceiling protection by preventing a lock holder from being preempted by any other thread*

- *Lock holders are required to yield the cpu if they prevented an actual preemption attempt*

- *The two varieties of non-preemptive mutexes protect against other pthreads in the same process and against any threads in the system*

Non-preemptive mutex performance



Provide Direct Access to Hardware

- *Memory map or allow programmed I/O access to hardware features*
- *Example: mmap() the Real-Time Clock*
    - *380ns clock_gettime() vs 2.4us from the kernel*
- *Example: mmap() a PCI device*
    - *full register level access from user space*

User Level Interrupts

- *Handle hardware interrupts in user space asynchronously*
- *Works great with memory mapped hardware*
- *31.2us to wake a waiting user thread*
- *13.6us to handle with a User Level Interrupt*

Direct I/O

- *Bypass the buffer cache by directly writing to and reading from user space to disk*

- *Eliminates memory copies and kernel overhead*

- *Allows the process to implement its own caching algorithm*

Inter Process Communication

- *Place message queues in shared memory areas*

- *2.9us to enqueue a message in shared memory*

- *6.7us to enqueue a message through the kernel*

Memory Map the Kernel

- *Provide useful per-thread data in special mapped pages of kernel memory*

- *Provides quick access to the current cpu ID, and scheduling and timing information*

- *Provides a window for the thread to indicate its signal mask and scheduling hints without entering the kernel*

Context Switches

- Some pthread implementations map multiple pthreads onto the same kernel thread

- Some things like signals become difficult but the benefit is that the kernel is less involved

- Many to one mapped pthreads can context switch in 1.6us vs 27.2us for one to one